A visualisation method for constructive solid geometry using polygon clipping

Anne Verroust

▶ **To cite this version:**

Anne Verroust. A visualisation method for constructive solid geometry using polygon clipping. RR-0461, INRIA. 1985. inria-00076093

HAL Id: inria-00076093
https://inria.hal.science/inria-00076093

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP105
78153 Le Chesnay Cedex
France

Tél. : (1) 39 63 55 11

Rapports de Recherche

N°461

A VISUALISATION METHOD FOR CONSTRUCTIVE SOLID GEOMETRY USING POLYGON CLIPPING

Anne VERROUST

Décembre 1985

A VISUALISATION METHOD FOR CONSTRUCTIVE SOLID GEOMETRY USING POLYGON CLIPPING

Anne VERROUST

Institut National de Recherche en Informatique,

Rocquencourt, BP 105,

78150 Le Chesnay

ABSTRACT: A hidden surface removal algorithm to visualize CSG solids in wire frame is proposed. This algorithm is an extension of the well-known ray-casting algorithm. It takes into account the coherence of the image to reduce the number of ray fired. For that, it uses a clipping polygon method, first introduced by Atherton and Weiler to resolve the problem of hidden line elimination for a set of polygonal objects.

RESUME: Un algorithme de visualisation d'objets décrits en CSG sans leurs faces cachées est présenté. Il tient compte de la cohérence de l'image pour réduire le nombre de rayons lancés dans la méthode de lancer de rayons (ray-tracing). Pour cela, il utilise une méthode de découpage de polygones dans le plan, introduite par Atherton et Weiler pour résoudre le problème d'élimination des lignes cachées dans le cas d'un ensemble d'objets polygonaux.

PAPIER RÉCUPÉRÉ ET RECYCLÉ

1. INTRODUCTION

One of the most current representations of 3D solids in CAD is to use CSG models: the solid is considered as the result of a boolean volumic combination of primitive solids; this combination is represented by a binary tree, the CSG tree of the solid (see section 4 for a more detailed definition). The CSG models are also commonly used as entries of ray-tracing algorithms [K] which produce realistic images with refractions, reflections and transparencies. In this paper, we are interested in rapid methods to generate images from CSG models. There are two general approaches to resolve the problem of visualisation [RV]:

1) Transform the CSG model into a boundary surface model, and use one of the hidden surface algorithms to visualize it [SSS].

2) Use the ray-casting technique [R]: fire a ray through each image picture element, and compute the first visible surface using the boolean rules and the CSG model.

The two approaches are time consuming:

In the first one, the computation of all the boundaries of the object is made, even if some parts of it are hidden. This computation can be useful when many views of the same CSG model are needed or when some additional computations (volume, etc....) are made.

In the second one, the whole screen is explored, without taking into account the coherence of the scene. Some authors have proposed solutions to minimize the cost of the

ray-casting technique in different ways:

Roth [R] proposes to use box enclosure to reduce the time cost of the ray fire process. This kinds of improvement does not work when the objects of the CSG tree mutually overlap. He also proposes, when the silhouette of the CSG object is only wanted, to sample the screen with rays and then locate the visible edges via a binary search. This may induce errors if the sample size is too large.

Atherton [A] uses coherence properties of the image to reduce the number of rays fired without introducing errors in the final image. His solution is adapted to scan line visualisation.

Bronsvoort, Jansen and Van Wijk [BJV] propose two kind of improvements to reduce the computation time: to use sampling like Roth but they know the disadvantages of the method (i.e. the imprecision of the resulting image), to use scan line enclosure instead of box enclosure and to simplify the CSG tree in an "active CSG tree" to reduce the boolean calculation of each ray. However, their improvements do not make the visualisation interactive.

Our goal is to make a CAD part of a ray-tracing program [NT]: we just want to visualize quickly the CSG model with hidden faces removed in wire frame before using the ray-tracing algorithm wich is expensive in computation time. The use of the ray-casting technique is well suited to us, as we just want a visualisation, and the structures are prepared for the ray-tracing program, but it has to be improved to

render the visualisation really interactive. We adopt, in our approach, Atherton's strategy [A], but as we want to visualize the CSG model in wire frame, our solution is relatively different.

Recall, to resolve surface visibility by using ray-cast, one proceeds as follows (see [R] for more details):

For each pixel in the screen,

- Compute the intersections between the visual ray and the solids composing the CSG tree,
- Sort the ray-solid intersections by Z-depth,
- Evaluate the first visible surface using the CSG tree.

Our CSG tree is composed of polygonal objects (the traditional primitives sphere, cylinder, etc ... are transformed into polygonal form). These objects are composed of facets. To minimize the number of rays fired, we want to obtain a subdivision of the screen in polygonal zones where the result of the ray-cast is constant.

The result of the ray-cast on any pixel p of the screen depends on:

- the solids encountered by the ray (i. e. the solids having p inside their projection on the screen),
- the respective order in Z-depth of the ray-solid intersections.

The subdivision of the screen must ensure that, in each polygonal zone belonging to the subdivision, if a projection of a facet overlap the zone, then the projection must cover the entire zone (i.e. the list of facets encountered by the

ray is a constant) and the respective order in z of the facets "visible" from the zone is constant in the entire zone. Thus, our visualisation algorithm consists in subdividing the screen into polygonal zone where the result of the ray-cast is a constant, fire a ray in each zone of the subdivision to find the visible facet, merge the adjacent zones having the same visible facets, and then obtain the image of the CSG solid with hidden faces removed.

The paper is organised as follows: We first describe, in section 2, the whole process of visualisation and then present in more details some particular aspects. In section 3, we describe the polygon clipping algorithm used to subdivide the screen. This algorithm computes the intersection, the union and the difference of two polygons that may be concave and may have holes in their contour. We define in that section the class of polygons considered. In section 4, we describe how we use the CSG tree to construct the first subdivision of the screen. To make that subdivision, we take into account the boolean combination coherence induced by the structure of CSG tree.

2. VISUALISATION ALGORITHM

We describe in this section the process followed to obtain an image of the SCG model with hidden facets removed. In all this section, the different phases of the process are illustrated by an example.

INPUT: A CSG tree of polygonal objects.
OUTPUT: A partition of the screen in polygonal zones corresponding to the image of the CSG tree with hidden lines removed.

A perspective orthographic transform is first made on all the objects of the CSG tree (see fig 1.). This transformation is made in such a way that the order in z is kept. We first work with the projections of the objects on the screen. A bounding rectangle is associated to each transformed object of the tree. We replace the objects by their rectangles associated in the CSG tree, subdivide the rectangles in smaller ones when some of them overlap, eliminate some of them using a rule induced by the CSG tree that we describe later, and then obtain a first partition of the screen in disjoint rectangles. While making this first partition, we associate to each constructed rectangle a list of facets belonging to projections of objects of the CSG tree. These facets are exactly the facets that may intersect the rectangle. We call these rectangles windows. In general, we call window any polygonal zone on the screen. At this step of the process, we have a partition P_0 of the screen in windows and for each window present in P_0, we know the facets that may overlap the window (see fig 2.).
We then want to obtain a new partition P_1 of the screen in disjoint windows such that, when a facet of any object of the CSG tree overlaps a window of P_1, this window is entirely included in the facet. To obtain P_1, each window

of P_0 is subdivided in smaller windows by recursively applying the following process:

At the beginning, $P_0 = P_1$ and for a window W of P_1, L_w and L'_w are exactly the lists of the facets associated to W. At the end of the process, P_1 contains windows W' such that $L_{w'}$ is the list of facets which totally overlap W' and $L'_{w'}$ is empty.

At each step, if all windows of P_1 have their second associated list empty,

then the process is terminated,

else, let W be a window of P_1 and F a facet present in L'_w. W is replaced in P_1 by the two subsets of disjoint windows: $\{W_1,\ldots, W_n\}$, $n>=0$, and $\{W'_1,\ldots, W'_m\}$, $m>=0$ and $m + n > 0$, where the W_i correspond to the sub-windows of W intersecting F (with $L_{wi} = L_w$ and $L'_{wi} = L'_w - \{ F \}$) and the W'_j correspond to the sub-windows of W having an empty intersection with F (with $L_{w'j} = L_w - \{ F \}$ and $L'_{w'j} = L'_w - \{ F \}$).

At the end of the process, we have a cover P_1 of the screen (see fig 3.) in disjoint windows $W_1,\ldots, W_k$ satisfying:

1) The list of facets associated to W_i, $0 < i < k+1$, is exactly the list of all the facets of the CSG tree totally overlaping this window and there is no other facet of the CSG tree intersecting it.

In general, the objects of the CSG tree may intersect each other in the space. These intersections do not appear in the projections of the facets on the screen. They can be viewed

as "portions of lines" cutting some of the windows of P_1, the lines corresponding to an intersection in the space of two facets. Thus, to obtain a partition P_2 of the screen where the intersections in the space appear, the windows of P_1 are subdivided in smaller windows when they contain two intersecting facets in their associated list. So, at the end of this process, we have a partition P_2 of the screen (see fig 4.) in windows satisfying 1), and:

2) the respective order in z of the facets present in the associated list of a window is constant for all the points inside it.

The screen is now partitionned in windows such that the result of the ray-cast is a constant inside any window. We now have the list P_2 of desired windows to execute the ray-cast:

For each window in P_2, a ray-cast is made in a point p inside this window. The CSG tree used for the ray-cast is the reduction of the initial CSG tree to the set of objects associated to the window (see in section 3. the explanation of the first partition of the screen for the definition of the reduction). The result of the ray-cast (i.e. the visible facet) is then associated to that window (see fig 5.).

Note that the partition of the screen that we have made may be too thin: it may contains adjacent windows having the same visible facet associated. Thus the windows having the same visible facet associated are combined by successively

making the union of two of them (see fig 6.).
At the end of that process the windows present in the partition form exactly the image in the screen of the CSG tree with hidden facets removed.

Example:

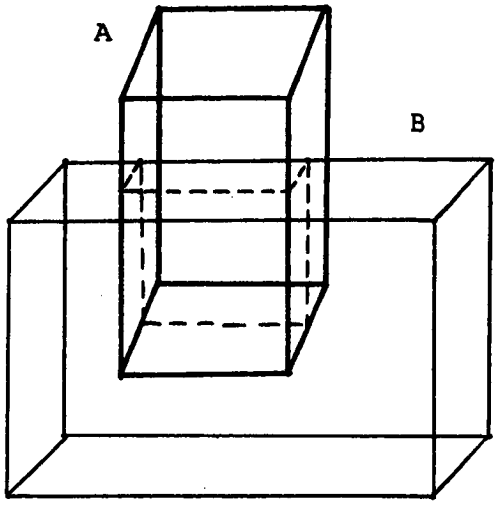


fig 1.

CSG tree: A - B

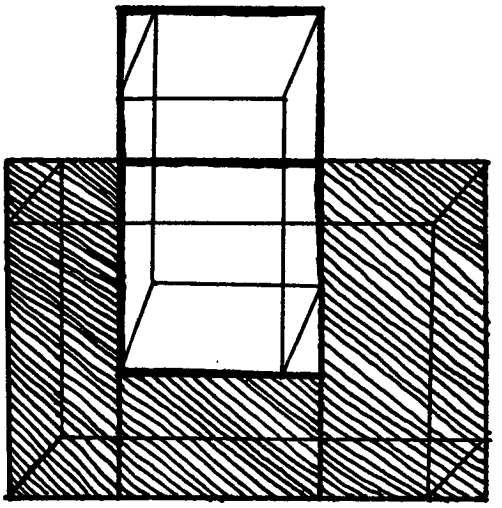


fig 2.

first partition in rectangles

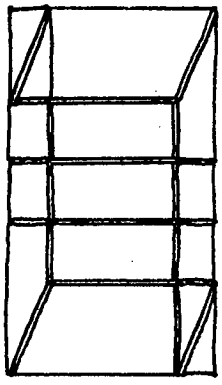


fig 3.

partition P_1

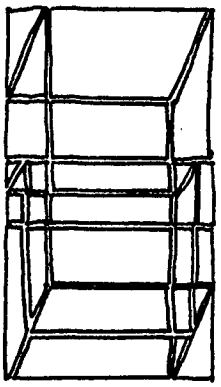


fig 4.

partition P_2

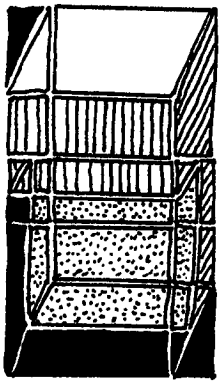


fig 5.

result of the ray cast

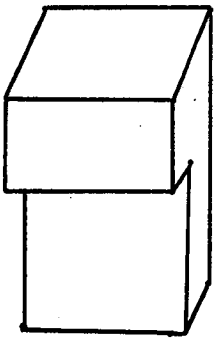


fig 6.

final image

All the operations of union, intersection or difference of two windows are computed by the same algorithm, the "polygon clipping algorithm". The principle of this algorithm was first introduced by Braid [B]. The general case was treated more precisely by Atherton and Weiler [AW].

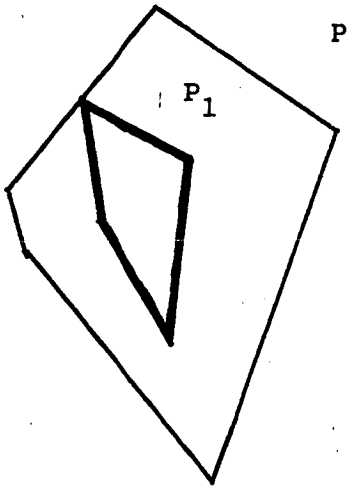
3. POLYGON CLIPPING ALGORITHM

We now present the "clipping algorithm", which computes the union, the intersection and the difference of two co-planar polygons. The principle of this algorithm can be found in [AW]. Our version details the particular cases that can be encountered during the computation.

INPUT: Two polygons;
OUTPUT: A list of polygons, results of either the union, the intersection or the difference of the input polygons.

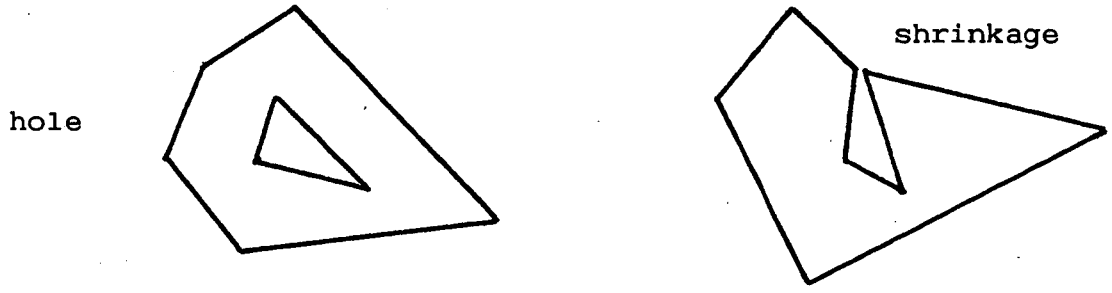
We first use this algorithm with, as inputs, a rectangle and a projection of a facet of an object of the CSG tree. As the original facets of the objects were simple planar polygons, the projection of the facets are either simple polygons or polygons reduced to a segment. The polygons reduced to a segment are eliminated while orienting the edges in a clockwise order.

When computing the difference of two simple polygons, the obtained polygons could be not simple:



$P_2 - P_1$
is not simple

We now introduce a class of polygons of interest that we call windows . We show that the results of the computation of the operations of difference, union and intersection of such polygons are still polygons belonging to this class. Informally, the windows are polygons that may have "holes" and "shrinkage" in their contour.



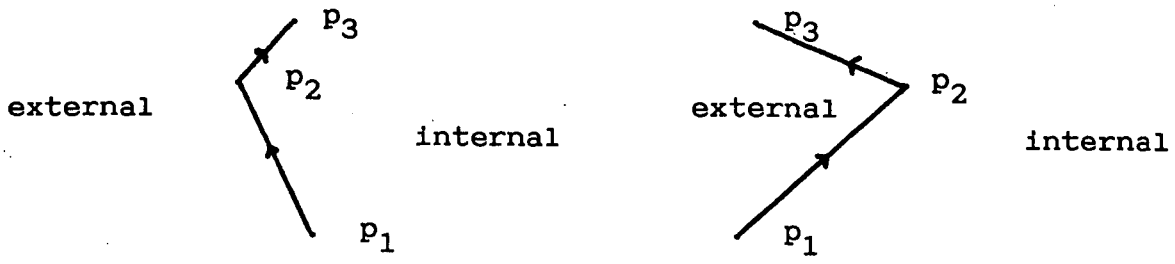
To define the notion of window, we first introduce some auxiliary concepts:

Given three distinct points p_1, p_2, p_3 such that $\overline{p_1p_2}$ and $\overline{p_2p_3}$ have just p_2 in common, a point p in the plane is said

internal to ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$) iff p belongs to the right part of the plane delimited by the two half lines directed by $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_2p_3}$ and ending in p_2.

external to ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$) otherwise []

This notion is natural if we consider that p_1, p_2 and p_3 are three consecutive vertices of a clockwise oriented polygon : then if the polygon is convex, when p is internal to the polygon, p is internal to ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$) . These two notions are equivalent in a neighbourhood of p_2.



We now define the notion of "shrinkage":

Given five distinct points p_1, p_2, p_3, p_4, p_5 such that $\overline{p_1p_2}$, $\overline{p_2p_3}$, $\overline{p_4p_2}$, $\overline{p_2p_5}$ have just p_2 in common, we say that ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$) and ($\overrightarrow{p_4p_2}$, $\overrightarrow{p_2p_5}$) do not cross each other iff:

p_1 and p_3 are both either external or internal to ($\overrightarrow{p_4p_2}$, $\overrightarrow{p_2p_5}$) and

p_4 and p_5 are both either external or internal to ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$)[]

Then if we consider that $\overline{p_1p_2}$, $\overline{p_2p_3}$, $\overline{p_4p_2}$ and $\overline{p_2p_5}$ are different edges of a polygon P, P has a "shrinkage" in p_2 when ($\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$) and ($\overrightarrow{p_4p_2}$, $\overrightarrow{p_2p_5}$) do not cross each other.

A window is composed of contours:

A contour $C(v_1,\ldots,v_p=v_1)$ of a polygon is defined as a cyclic graph.

It satisfies:

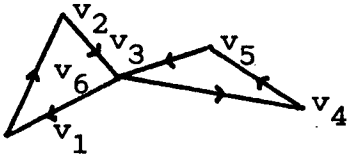
I. There is at least three non colinear points in $\{v_1,\ldots,v_{p-1}\}$.

II. There is no cross in C:

The common part of two different edges v_iv_{i+1} and v_jv_{j+1} is at most a vertex. If these two edges are non consecutive then $(\overrightarrow{v_{i-1}v_i}, \overrightarrow{v_iv_{i+1}})$ and $(\overrightarrow{v_{j-1}v_j}, \overrightarrow{v_jv_{j+1}})$ do not cross each other

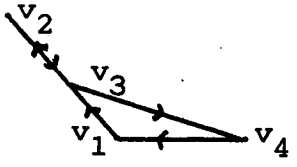
[]

For instance,



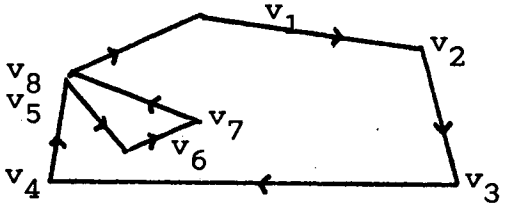
and

and



are not allowed for contours.

but:



is allowed.

As traditionnaly, a point p is said <u>inside</u> or <u>in the border</u> of $C(v_1,\ldots,v_p)$ iff either p belongs to an edge of C or there exists a half line issued from p having an odd number of intersections with the edges of C. Otherwise, it is said <u>outside</u> of $C(v_1,\ldots,v_p)$ []

A <u>window</u> W is defined as a list of contours $C_0,\ldots, C_n$ n>=0.

C_0 is the main contour of W. It represents the external boundary of W.

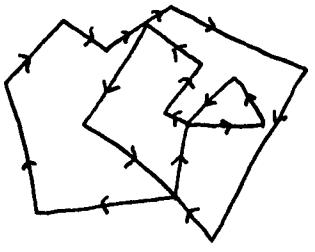
$C_1,\ldots, C_n$ are the hole contours of W. They represent the internal boundaries of W.

The vertices of C_0 are oriented in clockwise order and the vertices of C_i, $i>0$, are oriented in conterclockwise order. If $n>0$ then W must satisfy:

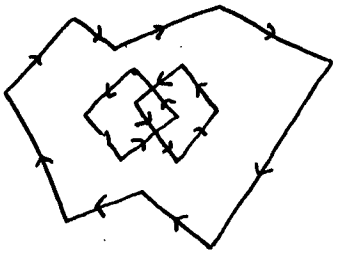
a) All the vertices of the hole contours are inside or in the border of C_0.

b) If C_i and C_j are two hole contours then all the vertices of C_i are outside or in the border of C_j.

c) Two edges of two different contours have at most one vertex in common []



window



not a window

Then a simple polygon is a particular case of a window: it has one contour, the main contour, and this contour satisfies the conditions I and II (because two non consecutive edges of a simple polygon do not intersect).

By definition of a window W,

_ The interior of W is:

{points inside C_0 } - $U_{i>0}$ {points inside C_i }

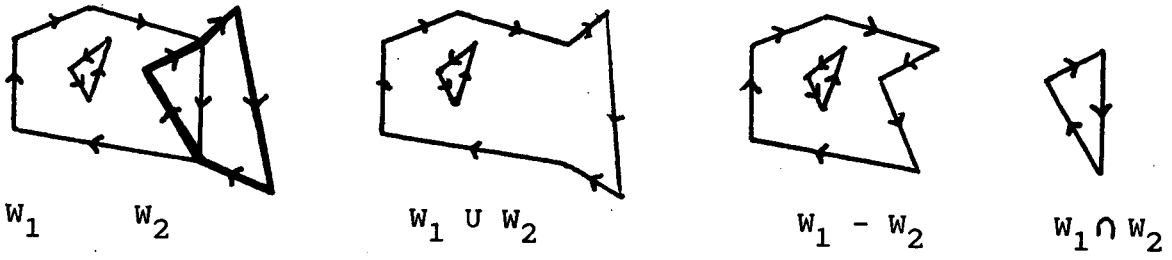
and it is never empty.

_ The choice of orientation for the contours ensure that the interior of the window is always to the right side of any edge of $C_0, \ldots, C_n$ []

We now can explain how the union, the intersection or the

difference of two windows are computed. We use as a basis the technique of Atherton and Weiler [AW].

Intuitively, given two windows W_1 and W_2, the contours corresponding to the union, intersection or difference of W_1 and W_2 are made of edges or part of edges of W_1 and W_2. They are either some of the contours of W_1 or W_2 or new contours. The intersection points between the two windows play an essential role in the construction of the new contours:



Thus as [AW] we process in two steps :

In a first step, that we call "preparation" step, the common points between the two windows, the equal contours, the contour having no intersection with all the other are marked. The "preparation step" is the same for all the operations: after this step any one of the three operations can be computed. In a second step, the new contours and the resulting windows are created.

Each window is represented by a list of contours. Each contour is a doubly linked chain of vertices so it is always possible to follow the contour in direct or in reverse order. To explain the whole process, the two windows will

be differentiated in: W_c, the clipping window (the window that will be used to "cut" the other, if we want to compute the difference, or any one of the two windows otherwise) and W_s, the subject window.

As [AW], during these two steps the following structures will be used:

A list of contours HOLIST which corresponds to the contours of the clipping or the subject window having no intersection with the other contours,

A list of couples of contours EQUALIST which correspond to the couples of equal contours,

Two lists of couples of vertices: INSIDE and OUTSIDE. The differentiation between the two input windows is used here: INSIDE (resp OUTSIDE) contains the vertices where the contours of the clipping window pass from the interior to the exterior (resp from the exterior to the interior) of the subject window.

The description of the "preparation step" follows:

The contours of the two windows are successively compared for intersection. The orientation of the different contours have been chosen in such a way that the processing of the main and the hole contours do not differ.

When a common point or a sequence of common sub-edges is encountered, the two lists INSIDE and OUTSIDE are used. We explain later in more details what is really done in these cases.

If one contour has no common part with all the other

contours, it is inserted in HOLIST.

When two contours are found to be equal, the couple is inserted in the list EQUALIST.

Now, let us explain in details how the insertions in the INSIDE and OUTSIDE lists are made.

$C=(v_1,\ldots, v_p)$ being a contour of W_c and $C'=(w_1,\ldots, w_q)$ a contour of W_s, suppose that the intersection between the two edges v_iv_{i+1} and w_jw_{j+1} is not empty. We consider the two cases of processing:

1. The intersection is reduced to a point p.
2. The intersection is equal to a sequence of sub-edges of C and C'.

case 1: The intersection is equal to a point p.

If p does not correspond to a vertex of C or C', a vertex equal to p is added in its place to the list of the vertices of C or C'. Let v and w be the vertices corresponding to p in C and C'.

Let v_o (resp w_o) be the first vertex before v (resp w) non equal to v in C (resp C') and v_f (resp w_f) be the first vertex after w (resp w) non equal to v in C (resp C').

There are two different cases to consider:

$(\overrightarrow{v_ov}, \overrightarrow{vv_f})$ and $(\overrightarrow{w_ow}, \overrightarrow{ww_f})$ do not cross each other or

$(\overrightarrow{v_ov}, \overrightarrow{vv_f})$ and $(\overrightarrow{w_ow}, \overrightarrow{ww_f})$ cross each other.

In the first case, there is a "shrinkage" between the two contours in v=w while in the second case there is a real

change between the locations of the two interiors of C and C'. As shrinkage is allowed in a contour or between two contours of windows, nothing is done in the first case.
When there is a cross in v, one of the two lists INSIDE or OUTSIDE is used:

_ If v_o is internal to ($\overrightarrow{w_o w}$, $\overrightarrow{w w_f}$), (v,w) is inserted in INSIDE,

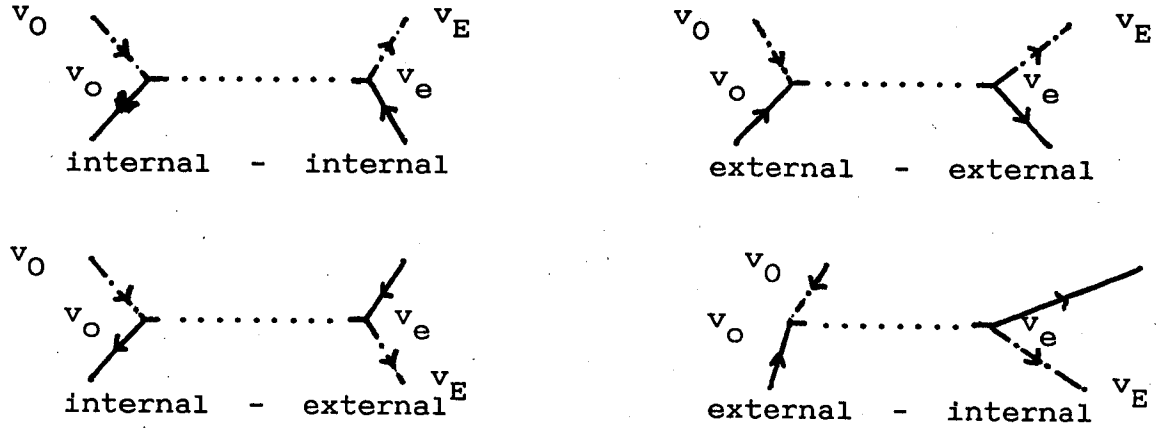
_ (v,w) is inserted in OUTSIDE in the contrary.

case 2: The intersection is equal to a sequence of sub-edges of C and C'. This case will be reduced in a proper manner to the first case.
The edges of the two contours are followed to find the "origin" and the "end" of the sequence of common sub-edges.
If a whole tour is made on the two windows during this research, the two contours are equal. Then (C, C') is inserted in EQUALIST.
If only one of the two contours have been entirely examined, then this contour is a sub-contour of the other. A vertex is added in this contour to artificially differentiate the origin and the end of the common sequence of sub-edges (cf. example in appendice).
Let v_o be the first vertex common in C (the "origin") and w_o the corresponding vertex in C'.Let v_e be the last vertex common in C (the "end") and w_e the corresponding vertex in C'. (C and C' have been modified in such a way that for each vertex in C between v_o and v_e there exists a corresponding vertex in C').

v_o and v_e are distinct vertices of C and w_o and w_e are distinct vertices of C'.

Let v_0 be the first vertex before v_o non equal to v_o in C and v_E be the first vertex after v_e non equal to v_e in C. v_0 and v_E can be either internal or external to C'.

Thus four cases of figure can be encountered:



The two first cases internal-internal and external-external are reduced to two intersections in a point, one at $v_o=w_o$ and the other at $v_e=w_e$. So,

in the case internal-internal (v_o,w_o) is inserted in the INSIDE list and (v_e,w_e) is inserted in the OUTSIDE list.

in the case external-external (v_o,w_o) is inserted in the OUTSIDE list and (v_e,w_e) is inserted in the INSIDE list.

external - external is viewed:

For the two cases external-internal and internal-external, a

new vertex v equal to v_e is inserted after v_e in C and a corresponding vertex w is inserted in C'.These two cases are then reduced to three intersections in a point: one a $v_o=w_o$, a second at $v_e=w_e$ and a third at v=w.

Thus,

in the case internal-external (v_o,w_o) and (v,w) are inserted in the INSIDE list and (v_e,w_e) is inserted in the OUTSIDE list.

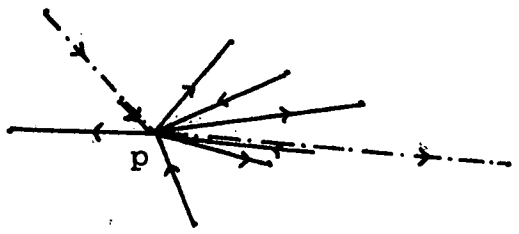
in the case external-internal (v_o,w_o) and (v,w) are inserted in the OUTSIDE list and (v_e,w_e) is inserted in the INSIDE list.

external - internal is viewed:

When reducing these four cases to intersections in a point, we have to remember for the future that the difference made between the edges is artificial. Thus the common edges are marked as "common".

Remark:

1) As "shrinkages" are allowed in windows, a point p can be the location of several different intersections:



Before inserting in INSIDE or OUTSIDE, we have to introduce a mechanism to differentiate all the intersections happening in p.

——— : W —·—·—: W'

Intuitively, as there is no cross in W and W', the solution is to distord in p the windows W and W' without adding new cross between W and W' and any cross in W or W'.

To simplify the problem, we suppose that p appears only once in the contours of W' (The mechanism can easily be extended to more general cases) .

We eliminate the intersections in a point without cross between W and W' because they do not modify the INSIDE or OUTSIDE lists. Then the intersections to examine are those where there is either a cross in p between the two contours or where p corresponds to an extremal vertex of a sequence of equal sub-edges between the two windows.

The problem occurs when:

p belongs to the contour C of W and p does not appear in other part of W.

C = (... v p v' ...)

S = {s | s vertex of W' equal to p such that
either the intersection in s is made with cross between W and W'
or s is an extremal point of a sequence of equal sub-edges between W and W'. }

|S| > 1.

Then, to all the elements s of S we associate a vertex v_s

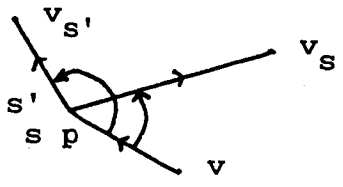
choosen as follows:

s belongs to a contour C' of W'. v_{so} is the first vertex before s non equal to s and v_{se} is the first vertex after s non equal to s in C'. As s belongs to S, $\{v_{so}, v_{se}\}$ contains a point v_s inside or in the border of W.

The vertices of s are ordered by << as follows:

$s << s'$ iff $(\overrightarrow{vp}, \overrightarrow{sv_s}) < (\overrightarrow{vp}, \overrightarrow{s'v_{s'}})$

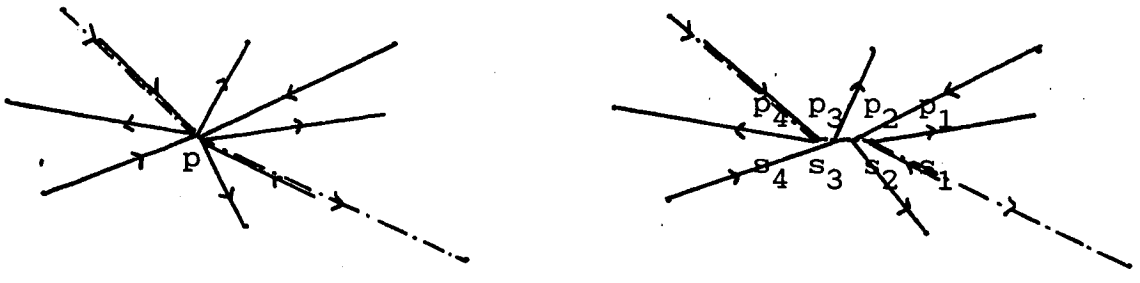
or, which is equivalent, $(\overrightarrow{sv_s}, \overrightarrow{s'v_{s'}}) > 0$.



$s << s'$

As W is a window, the order << is total in S. Let S be $\{s_1, \ldots, s_{|S|}\}$ with $s_i << s_j$ if $i < j$. $|S|-1$ vertices $p_2, \ldots, p_{|S|}$ are inserted after p in C. p is identified with p_1 and the p_i are matched with the s_i. Thus the different vertices are artificialy differenciated. Going back to the example:

we obtain:



[]

2) As all the points have been differentiated, at the end of the preparation step we have:

For any contour C of W_c or W_s, the ordered subsequence

$p_1, \ldots, p_k$ of C corresponding to vertices appearing in INSIDE or OUTSIDE is an alternate sequence of members of INSIDE and OUTSIDE []

Thus, at the end of this step we have all the necessary information in HOLIST, EQUALIST, INSIDE, OUTSIDE and in the two windows to perform the union the intersection or the difference of the two windows. We now explain how the resulting windows are formed.

As we have noticed before, the resulting windows have contours which:

either appear in HOLIST or EQUALIST (contours followed in reverse or in direct order),

or are new contours created from W_c and W_s using the two lists INSIDE and OUTSIDE.

For example, if $W_c=\{C_c\}$ and $W_s=\{C_s\}$ have elements in INSIDE and OUTSIDE, then the window corresponding to the union is made of:

The subsequences of C_c starting from a vertex of INSIDE to a vertex of OUTSIDE, and the subsequences of C_s starting from a vertex of OUTSIDE to a vertex of INSIDE.

Identically, the lists INSIDE and OUTSIDE can be used to describe the new contours corresponding to the union or the intersection.

In fact, the same algorithm is used to form the new contours for the three operations: LOOP(L, L', direct).

The parameters are defined by:

1) L and L'. { L, L' } = { INSIDE, OUTSIDE }. L is associated with W_c and L' with W_s.

2) direct. direct is true if the clipping window is followed in direct order and false if it is followed in reverse order.

The output part of LOOP is a list of new contours.

LOOP(L, L', direct):

1) If L is not empty
   then C <- 0 and go to 2),
   else exit

2) Choose a vertex v belonging to W_c and L such that, if direct is true, v is not the origin of a common edge in its contour and if direct is false, v is not the end of a common edge in its contour.
   If there is no vertex like that
   then go to 5),
   else note v as the initial vertex and go to 3).

3) Follow W_c in reverse order if direct is false and copy the vertices in C until a vertex v' appearing in L' is encountered (i.e. (v',w') belongs to L'). Delete (v',w') from L'.
   If w' is the initial vertex
   then add C to the list of new contours and go to 1),
   else go to 4).

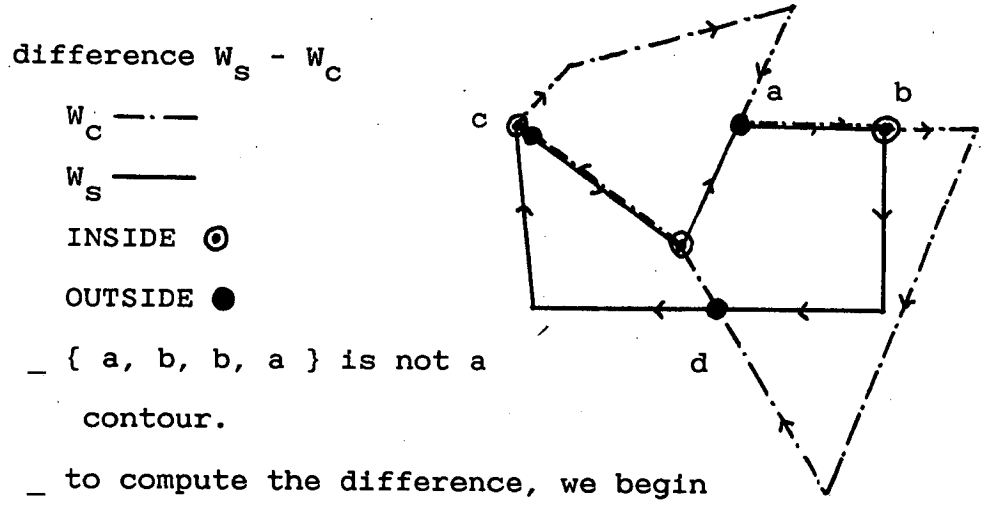
4) Follow W_s and copy the vertices in C until a vertex w appearing in L is encountered (i.e. (v,w) belongs to L). Delete (v,w) from L.
   If v is the initial vertex
   then add C to the list of new contours and go to 1),
   else go to 3).

5) Choose a vertex w belonging to W_s and L' such that w is not the origin of a common edge in its contour.
   If there is no vertex like that
   then exit (all the possible contours have been found),
   else note w as the initial vertex and go to 4).

end of the LOOP.

Remark:

The special choice of the first edge of the contour is due to the reduction made in the preparation step when considering common edges as a particular case of common points. Thus we must have the possibility to begin either on the subject or on the clipping window :



_ { a, b, b, a } is not a contour.

_ to compute the difference, we begin in d with the subject window (i.e. by step 5 of the LOOP). []

The computation of the new contours is done:

for the union of W_c and W_s: by LOOP(INSIDE, OUTSIDE, true),

for the intersection of W_c and W_s: by LOOP(OUTSIDE, INSIDE, true)

and for W_s minus W_c: by LOOP(INSIDE, OUTSIDE, false).

These new contours do not cross the contours belonging to the lists EQUALIST and HOLIST. There are at most "shrinkages" between them, which is allowed for different

contours of a same window.

To obtain the results we have also to test whether a contour is clockwise oriented or not and whether a contour is included or not in another contour. Since details of the construction are relatively straightforward, they are not given here.

This clipping algorithm is used several times in the whole process of visualisation:

1) to clip a window of the screen by a facet. The subject window is equal to the window of the screen and the clipping window is the facet. The subject window is partitionned in two subsets of windows: the windows resulting from the intersection process and the windows resulting from the difference (subject window minus clipping window).

2) to subdivide a window of the screen by a line corresponding to the intersection of two facets in the space. The subject window is the window of the screen. The clipping window is the window formed by one of the half of the screen delimited by the projection of the line. As in 1), the subject window is partitionned into two subsets of windows: the windows resulting from the intersection process and the windows resulting from the difference (subject window minus clipping window).

3) To gather two windows of the screen having the same visible facet associated. The two original windows play a symetric role, thus the clipping window is chosen

arbitrary. The computation is done only when the two windows are adjacent (INSIDE or EQUALIST non empty after the preparation step). The result is an unique window : the union of the subject and the clipping window.

4. FIRST PARTITION OF THE SCREEN INTO DISJOINT RECTANGLES.

We now describe how we use the CSG tree to build a partition of the screen into disjoint vertical rectangles covering the image on the screen of the CSG solid.

INPUT: A CSG tree of polygonal objects.
OUTPUT: A partition of the screen into disjoint rectangles, such that, the CSG tree reduced to the objects associated to the rectangles is not equal to the empty tree.

First, to recall, a CSG tree is a binary tree where

_ leaves are polygonal objects,

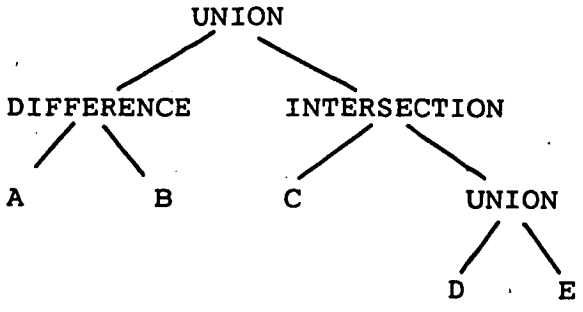
_ internal nodes are one of the three binary operations on solids:

UNION , A UNION B represents the volume formed by both A and B.

INTERSECTION , A INTERSECTION B represents the volume common to A and B.

DIFFERENCE , A DIFFERENCE B represents the volume of A non contained in B. In this case A is the left son (resp. B is the right son) of DIFFERENCE.

Each tree represents a solid S, resulting from volumic operations on all its leaves. This operation can be expressed in term of the three binary operations, UNION, DIFFERENCE and INTERSECTION by traversing the CSG tree in inorder:



represents:

((A - B) U (C ∩ (D U E)))

We want to construct a raw partition of the screen in rectangles using only:

- the structure of the CSG tree T representing the solid S,
- the vertical bounding rectangles of the projections of the polygonal objects.

This partition must cover the projection of the solid S on the screen. To do that, we interprete the volumic operations of UNION, INTERSECTION and DIFFERENCE into operations on "bounding" rectangles: given two rectangles R and R' bounding respectively o and o', then o INTERSECTION o' is bounded by R∩R', o UNION o' is bounded by R U R' and o DIFFERENCE o' is bounded by R (U, ∩ and - beeing the classical binary operations on polygons) .

As we want, for the future step of the visualisation algorithm, to know, for each rectangle of the resulting partition, the set of polygonal objects intersecting it

(i.e. to obtain couples $< R, O >$ where R is a vertical rectangle and O is the set of polygonal objects whose projection on the screen encounters R), we proceed as follows:

The initial CSG tree is traversed in postorder and

when a leaf o is encountered, a vertical bounding rectangle R is constructed and

{o} is the set of objects corresponding to R

(i.e. $< R,\{o\} >$ is created),

when an internal node is encountered, (the two sons have been yet encountered) then the two sets L_r and L_l of rectangles corresponding respectively to the right and to the left son are compared for intersection and reduced to a list L using the following rule:

(For the readability, we explain this rule in the case where L_r and L_l both contain only one element:

$L_r = \{ <R_r, O_r > \}$ and

$L_l = \{ < R_l, O_l > \}$. This rule can be extended in a simple manner to any set of rectangles)

Let R_i be the rectangle corresponding to the intersection of R_l and R_r. If R_i is not empty then the couples constructed form the set $T_i = \{ < R, O_l \cup O_r > \mid R$ belongs to $R_i \}$, else T_i is empty. Let S_r (resp. S_l) be a set of rectangles covering R_r and not R_l (resp. R_l and not R_r). The set of couples to consider are then

$T_r = \{ < R, O_r > \mid R$ belongs to $R_r \}$ and

$T_l = \{ < R, O_l > \mid R$ belongs to $R_l \}$. If R_r is empty then

T_r is the empty set. It is the same for T_l.

Then, if the internal node is equal to

UNION then $L = T_i \cup T_r \cup T_l$,

INTERSECTION then $L = T_i$,

DIFFERENCE then $L = T_i \cup T_l$.

Now we use the notion of "reduction of a CSG tree to a set of objects" to explain what has been done in this first partition:

Given a CSG tree T_0 and a set of objects O belonging to the tree, the reduction of T to O is made as follows:

At the begining, T is equal to T_0.

At each step, if T contains in its leaves an object o non included in O,

then o is extracted from T using SUBSTRACT(T,o) (described below),

else T is the reduction of the CSG tree.

The extraction of a node o belonging to a CSG tree T is recursively defined by SUBSTRACT(T,o):

If o is

a) the root of T, then SUBSTRACT(T,o) is equal to the empty tree,

b) the son of an UNION node. Let o' be the brother of o. Then SUBSTRACT(T,o) is equal to T where the father of o is replaced by o',

c) the son of an INTERSECTION node. Let be f the father of o. Then SUBSTRACT(T,o) = SUBSTRACT(T,f),

d) the right son of a DIFFERENCE node. Let o' be the brother of o. Then SUBSTRACT(T,o) is equal to T where the father of o is replaced by o',

e) the left son of a DIFFERENCE node. Let f be the father of o. Then SUBSTRACT(T,o) = SUBSTRACT(T,f).

Remarks:

1) The reduction of T to the set O of objects corresponds to the reduction of the expression associated to the tree replacing by the empty polygonal object the objects not belonging to O and then applying:

A INTERSECTION empty = empty, A UNION empty = A, A DIFFERENCE empty = A and empty DIFFERENCE A = empty.

2) While making the first partition, when an internal node is encountered in the traversal of the tree T, we eliminate the rectangles associated to a set of objects O such that, if T' is the sub-tree of T rooted by the node, the reduction of T' to O is the empty tree. Thus, as the last node encountered is the root of T, the partition obtained satisfies the condition.

3) If the reduction of T to O gives an empty result, then O do not contribute to form the "positive part " of the volume of S.

(it belongs to a sub-tree of T directed by a right son of a DIFFERENCE node or by a son of an INTERSECTION node).

In particular, when a rectangle of the screen is associated to a set O of objects such that the CSG tree reduced to O is empty, then we are sure that there is no

visible object in this part of the screen. Thus it can be eliminated without problem.

4) If a window of the screen has an associated list O of objects covering it, then using the remark 1) the ray cast inside this window can be either evaluated on the CSG tree T or on the CSG tree T', reduction of T to the set O. This kind of improvement corresponds to the use of the "active CSG tree" proposed by Bronsvoort, Jansen and Van Wijk [BJV].

5. CONCLUSION

We have described an algorithm to visualize CSG solids with hidden faces removed in wire frame. In this algorithm, we make a subdivision of the screen to reduce the number of rays of the traditionnal ray-cast technique. To obtain this subdivision, we use a polygon clipping algorithm, described in detail. This algorithm computes union, intersection and difference of two polygons belonging to a larger class of polygons than the class of simple polygons.

In this visualisation algorithm, we use several coherence techniques:

_ BOOLEAN COMBINATION COHERENCE. By making the first subdivision, we make a boolean simplification reasonning on the structure of the CSG tree.

_ FACE COHERENCE. There exists zones in the screen where we are sure that the result of the ray cast is a constant. These zones correspond to part of the screen where the

facets present in this part do not intersect each other and totally overlap this part. The subdivision obtained by our algorithm is composed of such zones.

These improvements do not alter the final image of the solid.

Some additional improvement could be made on this algorithm: find a method to mix more the ray-cast and the subdivision process to reduce the subdivision part of the algorithm. In fact, in our algorithm, all the projections of the facets are clipped together, even those which are finally hidden. This improvement could significantly reduce the computation time when the polygonal objects contain many faces.

This algorithm is under implementation in C language on a Perkin Elmer computer system. It is done at the Research Department of Image Synthesis of the INA (Institut National de l'Audiovisuel). The structure of the polygonal objects and a part of the visualisation have been made using a CAD program for polygonal objects designed by Alain Borenstein. This program will be used by Bruno Tezenas for its ray-tracing program.

ACKNOWLEDGEMENTS

I would like to acknowledge Etienne Beecker, Daniel Borenstein, Jean Charles Hourcade and Alain Nicolas of the Research Department of Image Synthesis of INA who gave me the oportunity to do this work in their department.

REFERENCES

[A] Atherthon, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry", Computer Graphics, Vol. 17, No. 3, July 1983, pp. 73-82.

[AW] Atherthon, P.R., Weiler, K., "Hidden Surface Removal using Polygon Aera Sorting", Computer graphics, Vol. 11, No. 2, Summer 1977, pp.214-222.

[B] Braid, I.C., "The synthesis of Solids Bounded by Many Faces", Communications of the ACM, Vol. 18, No. 4, April 1975, pp. 209-216.

[BJV] Bronsvoort, W.F., Jansen, F.W., Van Wijk, J.J., " Two Methods for Improving the Efficiency of Ray Casting in Solid Modelling", Computer Aided Design, Vol. 16, No. 1, January 1984, pp. 51-55.

[K] Kajiya, J.T., "Tutorial on Ray-Tracing", Siggraph 83.

[NT] Nicolas, A., Tezenas, B.,"An Illumination Model for Ray Tracing", Eurographics 85.

[R] Roth, S.D., "Ray Casting for Modeling Solids", Computer Graphics and Image Processing, No. 18, February 1982, pp. 109-144.

[RV] Requicha, A.A.G., Voelcker, H.B., "Solid Modeling: Current Status and Research Directions", IEEE Computer Graphics and Applications, October 1983, pp. 25-37.

[SSS] Sutherland, I.E., Sproull, R.F., and Schumacher, R.A., "A Characterisation of Ten Hidden-Surface Algorithms", ACM Computing Surveys, Vol. 6, No. 1, March 1974, pp. 1-55.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique